DIGITAL INVESTIGATION 5 (2008) S52–S57

available at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/diin

Digital Investigation

Forensic memory analysis: Files mapped in memory

R.B. van Baar, W. Alink, A.R. van Ballegooij*

Netherland Forensic Institute, 2497 GB, The Hague, Netherlands

Keywords:
Digital forensics
Volatile evidence
Memory mapped files
Microsoft Windows XP SP2

ABSTRACT

In this paper we describe a method for recovering files mapped in memory and to link mapped-file information process data. This information is forensically interesting, because it helps determine the origin and usage of the file and because it reduces the amount of unidentified data in a memory dump. To find mapped-file content, we apply several different techniques. Together, these techniques can identify approximately 25% of test memory dumps as being part of a memory-mapped file.

1. Introduction

In 2005, when the Digital Forensic Research Workshop (DFRWS) organized a memory challenge, contestants were encouraged to analyze the mechanics of the memory manager implemented in Windows 2000 (http://dfrws.org/2005/challenge/index.shtml). Before that time, physical memory of a computer was mainly captured to retrieve strings, e.g. passwords, IP addresses or e-mail addresses (Carvey, 2007). Going through these results manually is a tedious job; it is preferable to have a tool that can automatically identify relevant structures.

Modern Windows operating systems aggressively cache file data in memory. We therefore expect that a large part of the memory will be occupied by file data that is cached from the hard drive. Current forensic tools and techniques, however, do not take mapped-file information into account. This is unfortunate, because mapped-file information can be exploited in several ways. First, this information tells us something about the way a file was processed on a system. Second, it can provide information about recent activities on the system. Third, it helps reduce the amount of unidentified data in a memory dump.

The remainder of this paper is structured as follows. Section 2 describes file carving algorithms and the problems that occur when carving for files in memory dumps. Section 3 discusses the memory structures involved in managing mapped files. Section 4 explains the method developed for recovering mapped files from a memory dump. The next section contains experiments conducted, followed by Section 6, the related work. Section 7 shows the conclusions and the last section describes possible future work.

2. File carving in memory dumps

A method often used for reconstructing files from an image is carving. When carving for files, characteristic signatures are used to identify the start of a file. A popular carving program is Scalpel (http://www.digitalforensicssolutions.com/Scalpel/). Scalpel uses a linear carving technique, which is effective only for contiguous files. When a file is fragmented, linear carving algorithms fail to reconstruct the file and the file will be incomplete after the first fragment. Smart carving algorithms may be able to recover fragmented files, as described by Garfinkel (2007).

Unfortunately, carving is far less effective for recovering files from memory dumps. Many operating systems, including Windows, try to avoid fragmenting files on disk, which makes linear carving relatively effective. Data in memory, however, shows a high degree of fragmentation. We define a fragment as consecutive blocks of data from a file that are in consecutive pages of memory. It is possible for the memory manager to load only parts of a file into memory. If a block from

* *Corresponding author*.
E-mail address: ruud@holmes.nl (R.B. van Baar).
1742-2876/$ – see front matter
doi:10.1016/j.diin.2008.05.014

a memory-mapped file is not yet loaded into memory, this is also defined as one fragment.

Fig. 1 shows the average number of fragments in memory versus the number of blocks in a file. The data was gathered by matching blocks from files on a hard drive with pages in memory.[1] As shown in the graph, the fragmentation of files on the tested system approaches full fragmentation. This implies that almost no files can be reconstructed using linear carving methods. Even an algorithm that can recover fragmented files will fail to complete if a page is not yet loaded into memory. The method presented in this paper overcomes the need for file carving. This method interprets file-mapping related structures for Windows to reconstruct the order of the fragments. We are also able to link reconstructed files to the processes that have these files in use.

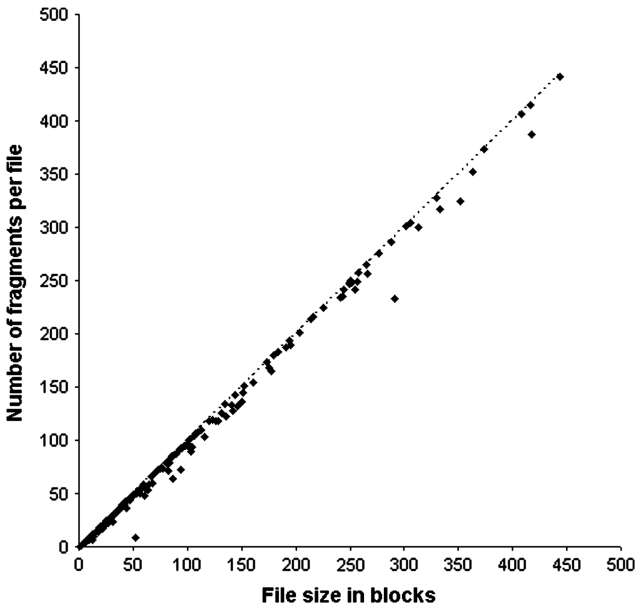

Fig. 1 – Number of blocks in a file set off against the average number of fragments per file in a memory dump. The dotted line indicates fully fragmented files.

3. Windows memory management

Just like a hard drive is divided into sectors, memory is divided into pages. On Intel architectures a page consists of 4096 bytes (4 K). A process is assigned pages through the memory manager. It can then use these pages to store data required by the process.

File mappings are administrated using a number of different data structures. An overview of these structures is shown in Fig. 2. These structures are allocated from memory pools. A memory pool is a dynamic memory area allocated by the kernel where it stores administrative structures. The type of a pool structure can be determined through pool tag (Schuster, 2006), a four byte magic number (e.g. Proc, Obtb, and MmCa) stored in the header of the structure.

Because we want to link mapped files to processes, the memory pool structures that administrate processes (Betz, http://dfrws.org/2005/challenge/betzReport.shtml; Garner, http://dfrws.org/2005/challenge/RossettoeCioccolato-Responses.pdf) are a good starting point. The process structure is further described by Schuster. The structure is identified via its pool tag Proc and contains pointers towards the Virtual Address Descriptor (VAD) Root (Dolan-Gavitt, 2007) and the *Object Table*.

The VAD Root is the root of the VAD tree. The VAD tree describes memory ranges in use by a process and enables reconstruction of a process his virtual address space. A node in this tree can have a number of different pool tags, depending on the type of Virtual Address Descriptor. Common tags are VadS, Vad and VadL. The latter two objects contain pointers to *Control Areas*, which are described below.

The Object Table is a list of private objects in use by a process and is identified through pool tag Obtb. Besides pointers to File objects (pool tag FILE), the Object Table also contains pointers to other objects, like registry key objects (pool tag Key) and event object (pool tag Evt).

The Control Area contains usage statistics of the mapped file and pointers to a File object, Page Table (pool tag MmSt) and Segment Object (pool tag MmSm). This object is identified through pool tag MmCa. A similar structure is the Control Area for Images (MmCi). The exact purpose of this structure is not clear, but likely it is used for administrating a copy of a number of pages from the data administrated by an MmCa structure. An extension of the Control Area is the Segment Object. This object contains, among other things, the file size of the mapped file.

File objects have a pointer back to a Control Area and a pointer to an Input/Output Name (pool tag IoNm), usually

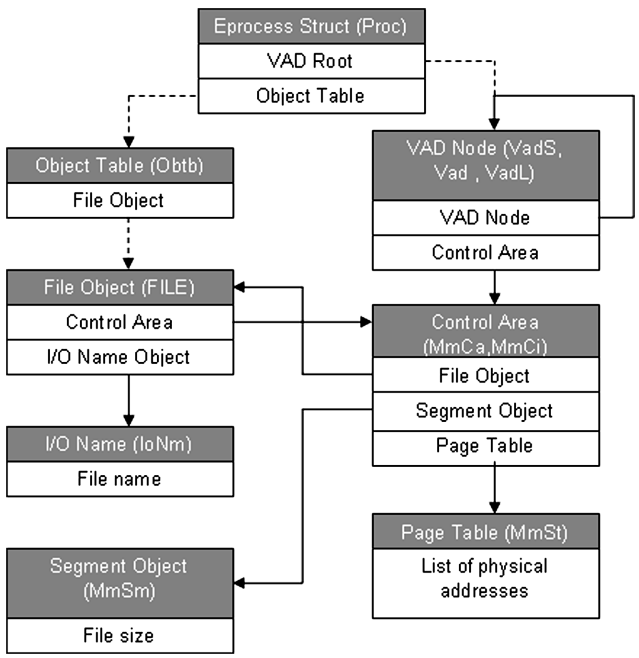

Fig. 2 – Overview of links between different memory structures related to mapped files. Dashed lines indicate pointers that are cleared when a process exits.

[1] Matching was done by calculating MD5 hashes of blocks and comparing the resulting hashes.

the file name and path on the hard drive or removable storage device. MmSt structures (Page Tables) contain lists of physical addresses that point to the pages containing file data. A full overview of pointers is shown in Fig. 2.

4. Mapped-file recovery

We have developed three methods for recovering files from memory. These methods, and a prototype system that implements them, are described below.

4.1. *Allocated file-mapping structures*

Using the carving algorithm implemented by Schuster in PTFinder we identify running, hidden and exited process structures (pool tag Proc). The running and hidden processes contain pointers to the VAD Root and Object Table. Processes that have exited have these pointers set to 0. By walking the VAD tree as described by Dolan-Gavitt (2007) we can identify shared files. We will refer to this method as *vadwalk*. Shared files are loaded by the kernel and can be accessed by any running process. By going through the Object Table it is possible to reconstruct private files; files that can only be accessed by the process that has mapped the file. This method will be referred to as *objecttables*.

4.2. *Unallocated file-mapping structures*

When file handles are closed by processes, file data may still be retained in memory. They cannot be linked to process structures, because the pointers to the mapped-file structures are set to zero when the handle is closed. This situation is similar to traces left behind by deleted files on a file system. The high degree of fragmentation of memory makes it harder to reconstruct the logical order of the pages, so determining this order by looking at the Page Table may make it possible to reconstruct files. If the related Control Area is still present, we can also link the file name to the file.

We can carve for the related structures of these files. Closed files can sometimes be recovered by carving for Control Area and Page Table structures (pool tags MmCa and MmSt). The first structure can give detailed information about the file like file name by linking to the IoNm (Input/Output Name) structure. It also contains a pointer towards the Page Table. Carving for Control Areas will be referred to as *mmca/mmci*.

Because it is possible that a Page Table is still present in memory after a Control Area has been overwritten, carving for Page Tables may be useful. Once the order of the pages has been reconstructed and the file extracted, it is sometimes possible to determine the file type by looking at header information. A commonly used tool for this is the Unix file command.

4.3. *Unidentified file pages*

It is still possible that pages previously used for storing mapped-file data have no more structures pointing to them. The file data is still present in the page and can be identified by matching the MD5 hash of the data from the page with hashes of 4 K blocks from files on the hard drive. Last blocks of files are padded to 4 K (memory page size) with zeroes before calculating the hash. Matching hashes is generic and does not require knowledge about the structure of the memory.

There are reasons not to use this method sooner. First, matching hashes requires access to the file system. Second, this technique does not link information about processes to files, and third, files that have been altered in memory will not be recognized.

4.4. *Prototype implementation*

Based on PTFinder and VADTools we have implemented a prototype program. This program currently supports memory dumps from Windows XP SP2 systems. However, minor alterations should provide support for other versions of Windows NT kernel based operating systems.

There are two types of output from the tool. First the tool writes the reconstructed files to an output folder, maintaining the directory structure as retrieved from the memory dump. Second, the tool creates an XML file that contains information per page. The XML document is designed to be plugged into other systems, like XIRAF (Alink et al., 2006). The XML file contains:

- Page information: page number, offset of the start and end of the page.
- File information: file name, path and sequence number of the page in the file.
- Process information: if a process can be linked to a page, the Process Identified (PID) is added as an element. It is possible to have multiple PIDs per page.
- Entropy: if no file can be linked to a page, the entropy is calculated to indicate the type of data in the page.

5. Experiments

To evaluate the behaviour of memory mapped files and the related administrative structures, three different experiments were conducted. The first experiment classifies the content of the mapped files. The second experiment evaluates the performance of the different methods and the third experiment looks at the relation between the time a system has been running and the amount of pages that can be identified as mapped files.

For the experiments around 90 different memory dumps have been created. Each dump was made after performing a variety of actions on the test system. These actions range from installing software and running some common office applications to browsing the internet and instant messaging. All memory dumps were created in a VMWare environment running Windows XP SP2 with 256 MB of memory, or, equivalently, 65,536 pages of 4096 bytes.

5.1. *Mapped-file content classification*

Table 1 shows common file extensions of files found in three different memory dumps. The first two images were used in

Table 1 – Common extensions on several memory dumps

DFRWS 2005 – 1		DFRWS 2005 – 2		Windows XP 256 MB	
.dll	202	.dll	211	.dll	298
.pnf	82	.exe	28	.gif	86
.exe	29	.mmf	11	.ttf	52
.mmf	11	.ttf	8	.ini	31
.png	11	.log	5	.jpg	30
.nls	9	.nls	5	.htm	29
.ttf	9		3	.exe	27
	8	.dat	3		22
.log	7	.evt	3	.png	22
.dat	4	.drv	3	.js	20
.sys	3	.fon	2	.css	17
.evt	3	.lmd	2	.lnk	15
.drv	3	.png	2	.dat	14
.apo	3	.apo	2	.txt	14
.lmd	2	Other	11	.nls	9
.ogg	2	Unknown	132	.acm	9
Other	11			.log	8
Unknown	272			Other	63

the DFRWS 2005 challenge (http://dfrws.org/2005/challenge/index.shtml). These were taken from a system running Windows 2000 with 128 MB of memory. The last memory dump was obtained from a system running Windows XP SP2 with 256 MB in a VMWare environment. This system was used for browsing the internet.

DLL files (shared libraries under Windows) are the most commonly identified file type. This is to be expected, because these usually are loaded by the kernel and can be used by any program for a range of common functions. The second most occurring extension differs depending on the way a system was used. The first system, which had been running for some time, had a lot of .pnf files. These files are created when an .inf file is run, most commonly during the installation of new software or hardware. On the second system, executable (.exe) files occur the second most. On the third system, the Windows XP machine, there are a lot of file types that are usually found in the Temporary Internet Files, e.g. .gif, .jpg, .htm and .js. The systems all have a number of .ttf files, TrueType Fonts. The Windows 2000 systems also have .mmf files, which in this case refers to files used by the installed McAfee VirusScan.

5.2. *Pages identified per method*

It is interesting to know which method generally identifies the largest number of pages. This experiment is based on the 90 memory dumps. The results are shown in Fig. 3.

The *objecttables* method results in the lowest number of identified pages. This method identifies the private files of a process which are not very common and only short-lived. The *vadwalk* method results in more pages identified than the previous method. This method identifies the shared files loaded by the kernel and used by different processes. The other two methods, *mmca/mmci* and *mmst*, carve the memory dump for occurrences of these objects. These methods result in the largest number of identified pages, because these objects are linked by both the Object Tables and the VAD tree. It also includes files that were previously in use but have been released. Scanning for Page Tables (MmSt structures) can yield more identified pages when the control structure (MmCa/MmCi) has been overwritten. It can also result in fewer identified pages because the carving algorithm for Page Tables is fairly conservative.

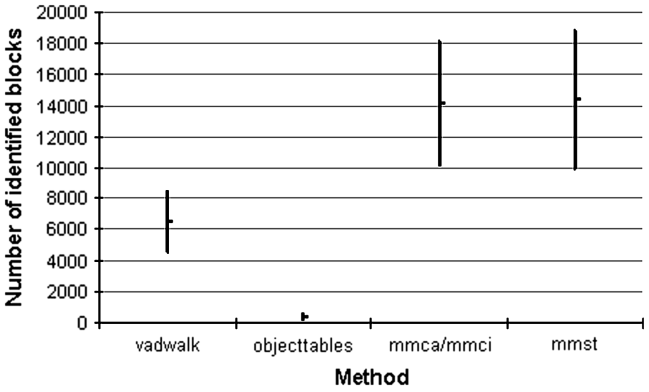

Fig. 3 – Pages identified per method, showing average, minimum and maximum number of identified blocks per method.

From Fig. 3, it follows that the average number of pages that can be identified using MmSt carving is approximately 25% of the data. Only the VAD Walk and Object Tables methods link process information. These combined methods link approximately 40% of the total identified pages to the related process or processes.

5.3. *Uptime versus number of pages identified*

In this experiment we determined the number of pages we were able to identify with varying uptimes of a system. Because processes are started and pages are filled with data, the expectation is that a longer uptime will result in fewer pages identified.

Fig. 4 shows the total number of pages identified and the number of empty pages in the image, versus the uptime of the system. We have defined a page as identified if either the page is empty or the contents of the page are found to be part of a mapped file by one of the proposed methods.

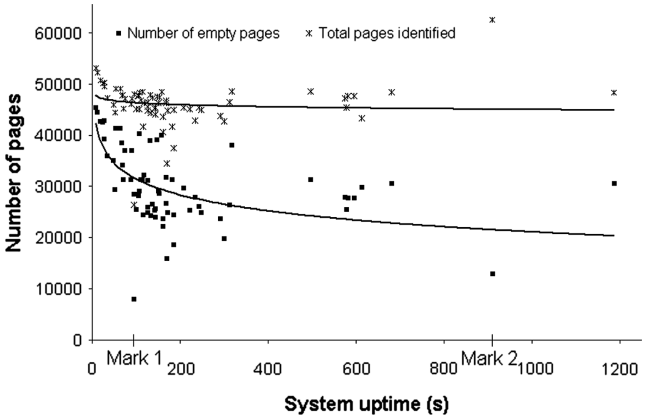

Fig. 4 – Uptime versus the number of pages identified.

Empty pages are defined as having an entropy (measure of randomness) lower than 0.5, which does not mean they are unused, but indicates a low information density. The general trend is that the longer the system is running, the fewer the pages are identified. The reason for this is that when a system has just booted a lot of pages are still empty. Soon after the system has started, programs are started and more pages are filled with data. The decline in total pages is smaller than the number of empty pages, because these empty pages are, among other things, filled with mapped files.

Two points in the graph have been marked. Mark 1 shows a valley in the graph in both the number of empty pages and number of identified pages. Just before the memory dump was created, the Windows Firewall was disabled and Skype was started. A possible explanation for this is that Skype encrypts its data in memory and decrypts it on the fly. Further research is needed to explain the results. Mark 2 shows a high number of total pages identified, but a low number of empty pages. Just before this memory dump was created, a dump of the physical memory was created and written to the physical drive of the system. A possible explanation is that this leads to many false positives, because the file containing the memory dump is then used to identify pages by matching 4 K hashes of data blocks from the file with pages of memory. This also needs further research.

6. Related work

Our mapped-file recovery techniques build on previous work by Schuster and Dolan-Gavitt (2007) We have implemented Schuster's algorithm for carving process structures. These process structures are linked to Dolan-Gavitt's method for walking the VAD tree to be able to reconstruct shared files.

The two most successful approaches of the DFRWS 2005 memory challenge (http://dfrws.org/2005/challenge/betzReport.shtml; http://dfrws.org/2005/challenge/RossettoeCioccolato-Responses.pdf) use allocated process structures to reconstruct the running process list and point out potentially harmful intrusions. Schuster presented a method to locate these process structures, including unallocated structures. This made clear that information about processes that have exited is not removed immediately from memory. Schuster (2006) describes how pool allocations work and how to locate structures in memory pools. Memory pools were discussed in Section 3.

Dolan-Gavitt (2007) introduced a method to walk the Virtual Address Descriptor (VAD) tree. Also in 2007 AAron Walters and Nick Petroni released Volatility, a collection of tools that provides insight into digital traces contained in a memory dump. The main focus is offering a replacement for incident response tools by retrieving relevant data from a memory dump, e.g. system time and date and open file handles. It currently also supports the VAD Walk algorithm and carving for process structures. Retrieving this information from a memory dump reduces the intrusiveness of incident response tools.

Fragmentation of data in memory has not received a lot of attention. For data on hard drives Garfinkel (2007) has researched file fragmentation. His research shows that only a fraction of the files on a hard drive is fragmented. This may explain the limited attention for reconstructing fragmented files in research related to carving. Yet reconstructing fragmented files is a necessity when searching for files in memory due to the high degree of fragmentation.

7. Conclusions

As shown, it is possible to retrieve mapped files from memory. From the memory dumps created for the experiments around 25% of the pages in the dump could be identified as part of a mapped file. Of these identified pages approximately 40% could be linked to the relating process or processes. Looking at the different methods, the general trend is that the less information we want to link to a page, e.g. process information or file name, the more pages can be identified. This is caused by unallocated structures that can still be identified, but not linked to other information in the memory dump.

By identifying mapped-file data in memory dumps and linking this information to process structures we obtain information about the origin and usage of a file. It also reduces the amount of unknown data in a memory dump.

8. Future work

For future implementations it should be possible to add more versions of Windows. Some small adjustments in the code allowed the program to work with the Windows 2000 memory dumps and we expect that similar adjustments should allow for the program to work on Windows Vista memory dumps.

It may be possible to extend the tools to include the page file. This way, pages that are swapped out can be added to the analysis (Kornblum, 2007). Note, however, that Farmer and Venema (2005) suggested it is unlikely to find mapped files in the page file.

The MmCi structure is currently supported by our prototype. The structure is similar to the MmCa structure, however, it is not entirely clear what the function of the structure is. Further research into the structure may be needed to determine its function.

REFERENCES

Alink W, Bhoedjang RAF, Boncz PA, de Vries AP. XIRAF – XML-based indexing and querying for digital forensics. Digital Investigation 2006;3S:S50–8.

Carvey Harlan. Windows forensic analysis. Syngress; 2007 [chapter 3].

Dolan-Gavitt Brendan. The VAD tree: a process-eye view of physical memory. Digital Investigation 2007;4S:S62–4.

Farmer Dan, Venema Wietse. Forensic discovery. Third Printing. Pearson Education; 2005 [chapter 8.15].

Garfinkel Simson L. Carving contiguous and fragmented files with fast object validation. Digital Investigation 2007;4S:S1–12.

Kornblum Jesse D. Using every part of the buffalo in Windows memory analysis. Digital Investigation 2007;4:24–9.

Schuster Andreas. Searching for processes and threads in Microsoft Windows memory dumps. Digital Investigation 2006;3S:S10–6.

Schuster Andreas. Pool allocations in windows memory forensics. IMF, http://www.gi-ev.de/fachbereiche/sicherheit/fg/sidar/imf/imf2006/23_Schuster-PoolAllocations.pdf; 2006.

Ruud van Baar holds a Masters in Grid Computing with a Forensic Science Minor from the University of Amsterdam. After completing his internship at the Netherlands Forensic Institute in 2007, he started working there as a digital forensic scientist.

Wouter Alink received the MSc degree in Computer Science from the University of Twente in 2005. After finishing his master's thesis titled 'XIRAF' at the Mathematical and Computer Science Research Institute (CWI) in 2005, he joined the Digital Technology Department of the Netherlands Forensic Institute, where he is currently employed as a research fellow.

His main research interests comprise XML database architecture in general and multi-dimensional markup.

Alex van Ballegooij received the MSc degree in computer science from the Vrije Universiteit in Amsterdam. After doing PhD research in the field of database technology at the Center for Mathematics and Computer Science in Amsterdam, he joined the Netherlands Forensic Institute in 2007. He is currently employed as a Forensic Scientist doing both research and case work.